Untitled

January 22, 2018

In [111]:
```
%pylab inline

import pymc3 as pm
from pymc3.backends import SQLite
import triangle
import pandas as pd
```

```
Populating the interactive namespace from numpy and matplotlib
```

1 Los datos

In [112]:
```
dataHz = np.loadtxt('Hz_all.dat') #We read the data. They are in the archive Hz_all.da
```

In [113]:
```
'''We save our different data in new variables.
z ---> Redshift
observations ---> Our observations
errors ---> The errors of our observations
'''
z = dataHz[:,0]
observations = dataHz[:,1]
errors = dataHz[:,2]
```

In [114]:
```
figsize = (8, 6)                          # definimos el tamaño de nuestra figura
dpi = 300                                  # dots per inch

rcParams['font.size'] = 10               # establecemos el tamaño de la fuente
rcParams['lines.linewidth'] = 1            # el grosor de las líneas
rcParams['mathtext.fontset'] = 'cm'        # y el tipo de fuente en LaTex

fig1 = figure(figsize=figsize, dpi=dpi)  # definimos la figura


plt.errorbar(z, observations, errors,      #plt.errorbar(x, y, error_y)
             xerr=None,
             color='red', marker='o', ls='None',
             elinewidth =1, capsize=3, capthick = 1,
```

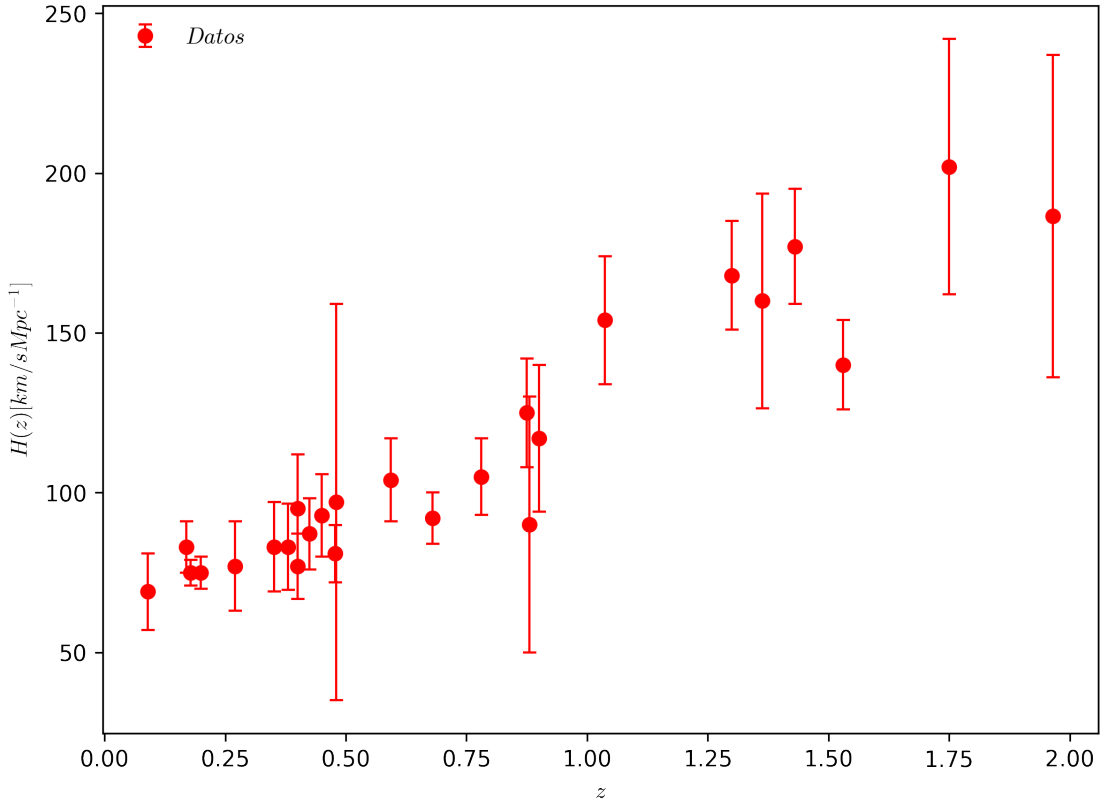
```
                    label='$Datos$')


        plt.legend(loc='best', frameon=False)     #pon la legenda donde mejor quede sin marco

        xlabel(r'$z$')
        ylabel(r'$H(z) [km/s Mpc^{-1}]$')


        #savefig('data_plot.pdf', bbox_inches='tight') #guardar la salida como un archivo pdf,
```

Out[114]: <matplotlib.text.Text at 0x140b6f90>



2 Definimos el problema y el modelo teórico

Consideremos la extensión más simple del modelo de LCDM (materia oscura fría con constante cosmológica Λ). Consideraremos que Λ no es una constante, pero sigue una ecuación de estado de la forma $p = \omega\rho$ con $\omega \equiv Cte$.

Si consideramos un universo plano, se cumple que $\Omega_m + \Omega_{DE} = 1 \Rightarrow \Omega_{DE} = 1 - \Omega_m$

Digamos que utilizamos las "standar candles" (SNIa) como observaciones para medir a que velocidad se expande el universo para cierta etapa de la historia del cosmos.

De cosmología sabemos que el parámetro de Hubble evoluciona en función del Redshift y el contenido de la materia como

2.0.1 $H(z) = H_0\sqrt{\Omega_m(1+z)^3+\Omega_\Lambda}$

donde H_0 es el valor del parámetro de Huble en nuestra época. Los parámetros que vamos a estimar serán el valor de H_0, ω y Ω_m.

```
In [115]: def hubblefunc(z, w, H0, OmegaM):
              '''
              This function calculates the theoretical value for the Hubble function,
              this is, the expansion rate of the Universe in terms of redshift

              w: Dark Energy Equation of State. w=-1 for a Cosmological Constant
              H0: present value of Hubble constant
              OmegaM: fractional matter density

              '''
              matter_contribution = OmegaM *(1 + z)**3

              DE_contribution = (1 - OmegaM) * (1 + z)**(3 * (1 + w))

              Ez = np.sqrt(matter_contribution + DE_contribution)

              return H0*Ez
```

3 Hacemos la inferencia de parámetros

Vamos a considerar un Likelihood Gaussiano, i.e. $L(\vec{\theta}) = exp\left[-\frac{1}{2}\sum_i^{No.Obs}\frac{(H(z_i)_{obs}-H(z_i,w=-1,H_0,\Omega_m))^2}{\sigma_i^2}\right]$ ##### donde $\vec{\theta} = \{H_0, M\}$. ##### Vamos a suponer que no tenemos información de los parámetros, salvo sus cotas límite. Digamos que $\Omega_m \in [0.1, 1]$ y $H_0 \in [10, 100]$. Por la ignorancia de nuestros parámetros, un buen prior que podemos considerar para estos es tomar un prior uniforme ##### $\Omega_m \sim U[0.1, 1]$ ##### $H_0 \sim U[10, 100]$

```
In [116]: '''Now we are going to give our model to PyMC'''
          with pm.Model() as model:
              OmegaM = pm.Uniform('OmegaM',lower=0.1,upper=1.0)
              H0 = pm.Uniform('H0',lower=10.0,upper=100.0)
              H_z = hubblefunc(z, -1, H0, OmegaM)

              Lik = pm.Normal('Lik',mu=H_z,sd=(2**(1/2))*errors,observed=observations)
```

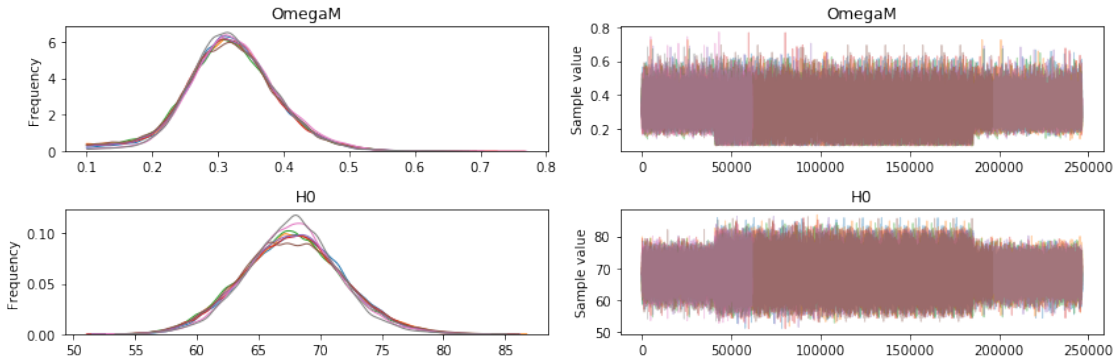
```
In [117]: #We specify the number of iterations
          niter=50000
```

```
        with model:
            start = pm.find_MAP()
            step = pm.Metropolis()
            db = SQLite('trace.db')
            trace = pm.sample(niter, trace=db, step=step, start=start,njobs=5, random_seed=123
```

```
In [118]: start
```

```
Out[118]: {'H0': array(68.21080972),
           'H0_interval__': array(0.60494477),
           'OmegaM': array(0.31849471),
           'OmegaM_interval__': array(-1.13754222)}
```

```
In [119]: with model:
              tracee = pm.backends.sqlite.load('trace.db')
              pm.traceplot(tracee, varnames=['OmegaM','H0'])
```




```
In [139]: t = trace[niter//2:]
          t['OmegaM'].shape
          t['H0'].shape
```

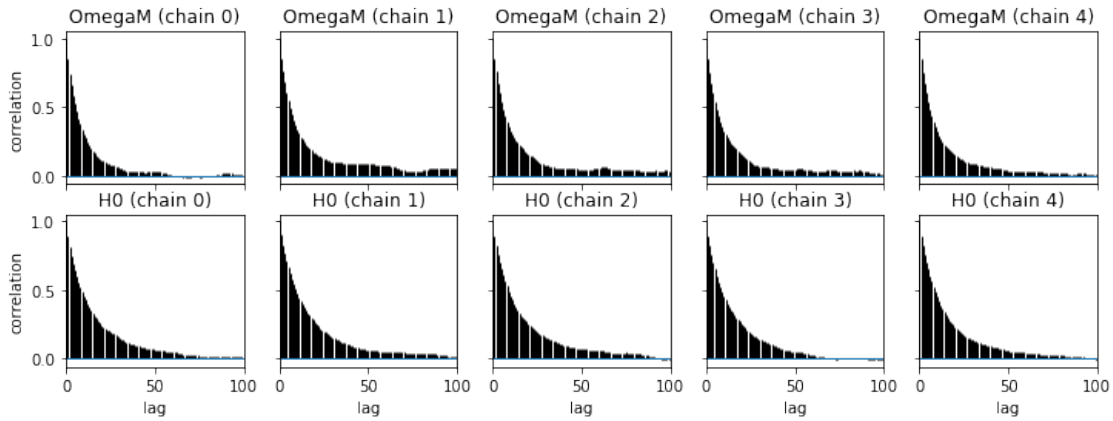
```
Out[139]: (1075000,)
```

```
In [121]: OmegaM = trace.get_values('OmegaM', burn=niter//2, combine=True, chains=[0,2])
          OmegaM.shape

          H0 = trace.get_values('H0', burn=niter//2, combine=True, chains=[0,2])
          H0.shape
```

```
Out[121]: (430000,)
```

```
In [122]: pm.autocorrplot(t, varnames=['OmegaM','H0'])
          pass
```



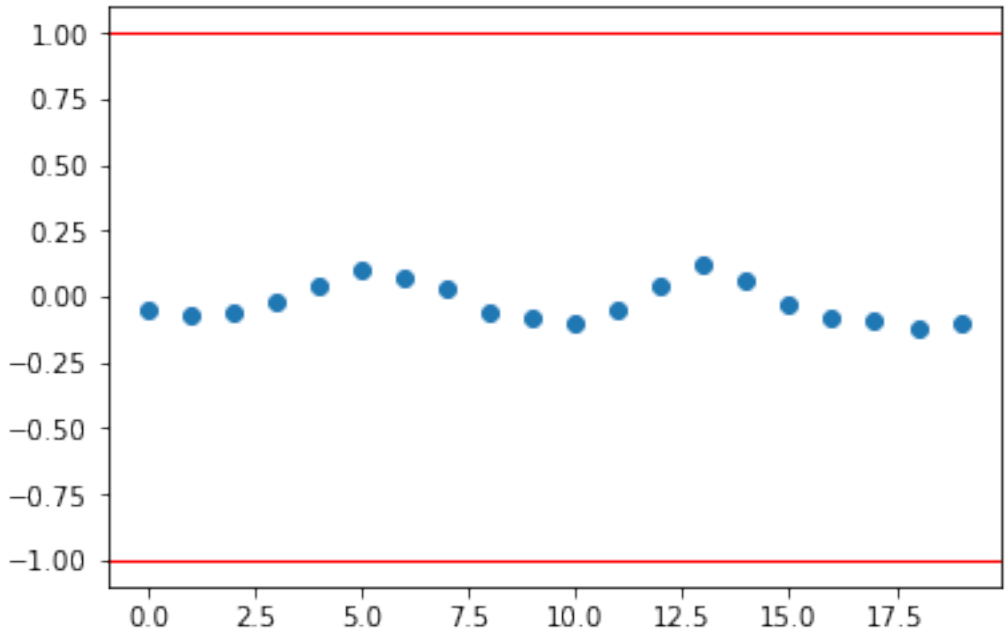

```
In [123]: pm.effective_n(t)
```

```
Out[123]: {'H0': 34391.0, 'OmegaM': 7537.0}
```

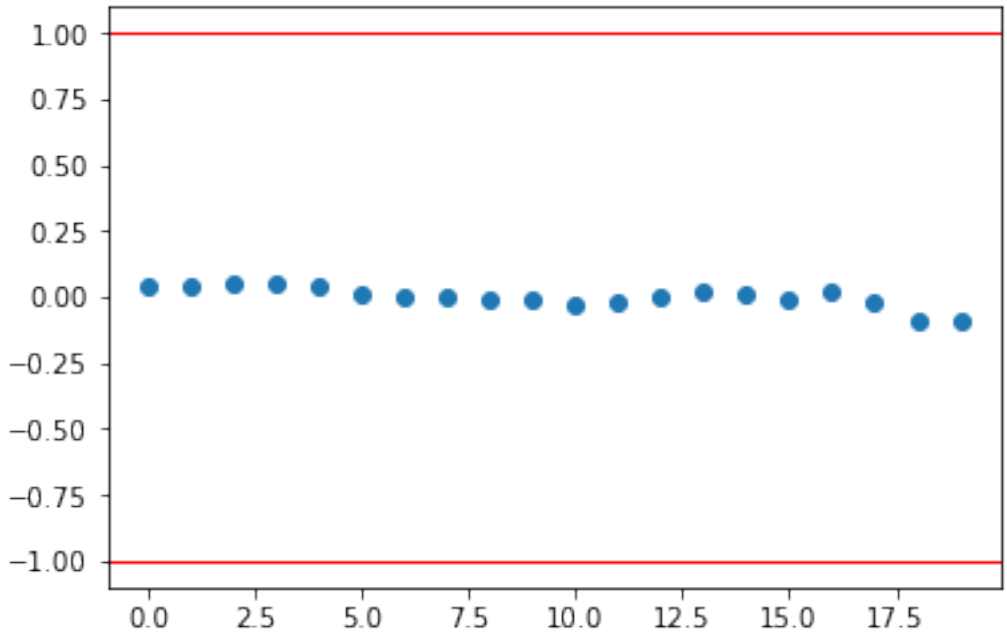
```
In [124]: pm.gelman_rubin(t)
```

```
Out[124]: {'H0': 1.0004853292008946, 'OmegaM': 1.0000401305986388}
```

```
In [125]: plt.plot(pm.geweke(t['OmegaM'])[:,1], 'o')
          plt.axhline(1, c='red')
          plt.axhline(-1, c='red')
          plt.gca().margins(0.05)
          pass
```



In [126]:
```
plt.plot(pm.geweke(t['H0'])[:,1], 'o')
plt.axhline(1, c='red')
plt.axhline(-1, c='red')
plt.gca().margins(0.05)
pass
```

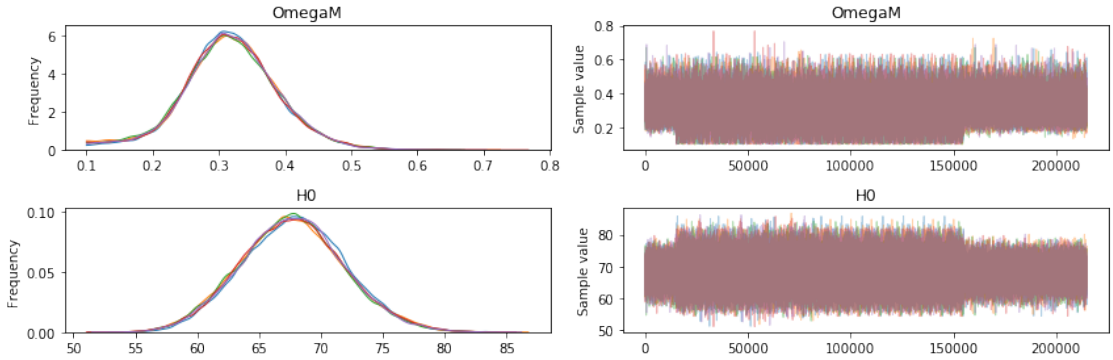


In [127]:
```
pm.summary(t, varnames=['OmegaM','H0'])
```

Out[127]:

	mean	sd	mc_error	hpd_2.5	hpd_97.5	n_eff	Rhat
OmegaM	0.317201	0.072718	0.001001	0.164740	0.469017	7537.0	1.000040
H0	67.691635	4.201709	0.025852	59.541644	76.058639	34391.0	1.000485

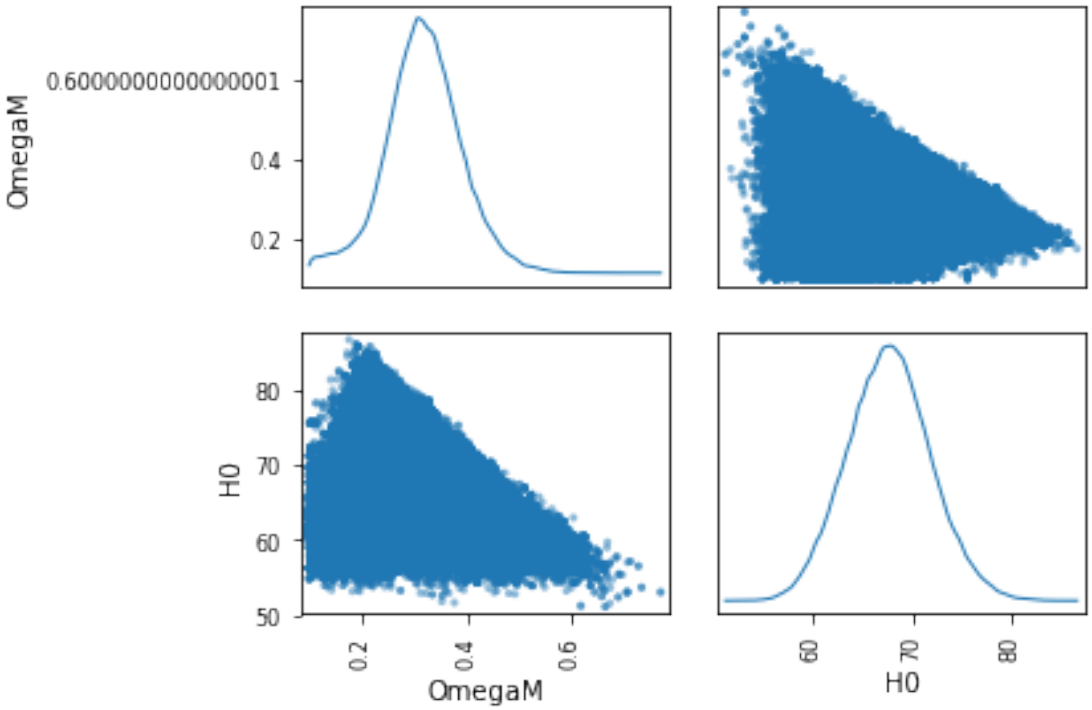
In [128]:
```
pm.traceplot(t, varnames=['OmegaM','H0'])
pass
```



```
In [144]: df_trace = pm.trace_to_dataframe(trace[niter//2:])
          pd.scatter_matrix(df_trace.ix[-niter//2:, ['OmegaM', 'H0']], diagonal='kde')
          plt.tight_layout()
          plt.show()
          pass
```

```
/home/enriques/Downloads/yes/lib/python2.7/site-packages/ipykernel/__main__.py:2: DeprecationWar
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  from ipykernel import kernelapp as app
/home/enriques/Downloads/yes/lib/python2.7/site-packages/ipykernel/__main__.py:2: FutureWarning:
  from ipykernel import kernelapp as app
```




```
In [165]: plot(OmegaM, H0,
               linestyle='none', marker='o', color='grey', mec='grey',
          alpha=.05, label='Posterior', zorder=-100)
```

```
import scipy.stats
gkde = scipy.stats.gaussian_kde([OmegaM, H0])
x,y = mgrid[0.:1.:.005, 10.:100.:.5]
z = array(gkde.evaluate([x.flatten(),y.flatten()])).reshape(x.shape)
contourf(x, y, z,3, linewidths=1, alpha=.5, cmap='Greys')
plt.colorbar();

ylabel(r'$H0$', fontsize=18, rotation=0)
xlabel(r'$\Omega_m$', fontsize=18)
legend()
axis([0.2, 0.6, 60., 80.])
savefig('param_dist.png')
```